Mass Provisioning of Nordic BLE Mesh Nodes

|  |  |  |
| --- | --- | --- |
| **Document version** | **Author** | **Comment** |
| Oct13-2019 | MQ | Initial document for Mesh SDK 3.2.0. |

# Purpose

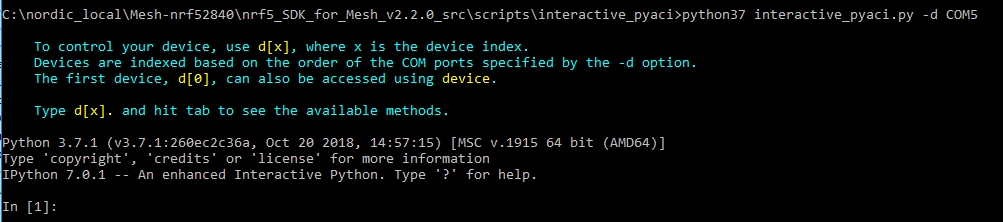
When creating a mesh network with many nodes, it can take a while to provision each node, especially using PyACI. An alternative is to use a static provisioner that provisions the nodes automatically but that requires a dedicated node that is used for this purpose.

This document outlines the use of a Python script, meshfwpatch.py, that can take as input firmware that has been ripped (read back from node’s flash and saved as an Intel Hex file using nrfjprog) from a provisioned node, update the device address and the device key in that firmware, write the updated firmware to a new file, and, finally, update the mesh database file. This allows for the quick generation of firmware that can be flashed onto nodes. The nodes are, therefore, pre-provisioned from the start automatically. It is only required to manually provision one node and then that node’s firmware can be ripped and new copies of firmware generated using this script for any number of other nodes.

# Procedure

In this procedure, we will provision a node using PyACI, rip its firmware, and then use the meshfwpatch.py script to generate new firmware. We will then flash this firmware to a new node and observe it as coming up provisioned. This process can be extended to any number of nodes.

1. Connect a nRF52-DK to the PC via USB cable and launch the Segger Embedded Studio (SES) with the project in <Mesh SDK 3.2.0>/examples/serial/ directory.
2. Compile, flash, and run the serial firmware on the nRF52-DK.
3. Follow instructions [here](https://www.nordicsemi.com/en/DocLib/Content/SDK_Doc/Mesh_SDK/v3-1-0/md_scripts_interactive_pyaci_README) to install Python 3 and launch the PyACI tool under <Mesh SDK 3.2.0>/scripts/interactive\_pyaci/:



1. Connect another nRF52-DK to the PC via USB cable and launch the Segger Embedded Studio (SES) with the project in <Mesh SDK 3.2.0>/examples/light\_switch/server/ directory.
2. Compile, flash, and run the light switch server firmware on the nRF52-DK.
3. We need to start with a fresh JSON file which will hold the database of the mesh network. This file resides in the <Mesh SDK 3.2.0>/scripts/interactive\_pyaci/database/ directory. Note that this file can be edited by humans since it is human-readable. Copy “example\_database.json.backup” to “example\_database.json” in the same folder. Overwrite, if needed. This will be a necessary step each time only when provisioning a brand-new network.
4. Now we will provision the light switch server node. In PyACI shell, type in the highlighted input:

In **[**1**]:** db **=** MeshDB**(**"database/example\_database.json"**)**

In **[**4**]:** p**.**scan\_start**()**

In **[**5**]:** p**.**scan\_stop**()**

In **[**6**]:** p**.**provision**(**name**=**"Light bulb"**)**

In **[**7**]:** cc **=** ConfigurationClient**(**db**)**

In **[**8**]:** cc**.**force\_segmented **=** **True**

In **[**9**]:** device**.**model\_add**(**cc**)**

In **[**10**]:** cc**.**publish\_set**(**8**,** 0**)**

In **[**11**]:** cc**.**composition\_data\_get**()**

In **[**12**]:** cc**.**appkey\_add**(**0**)**

In **[**13**]:** cc**.**appkey\_add**(**1**)**

In **[**14**]:** cc**.**model\_app\_bind**(**db**.**nodes**[**0**].**unicast\_address**,** 0**,** mt**.**ModelId**(**0x1000**))**

This should provision the node.

1. Launch a command line window, go the <Mesh SDK 3.2.0>/scripts/interactive\_pyaci/ directory, and issue the following command to rip the firmware from the provisioned light switch server node:

nrfjprog --family nrf52 --readcode dump\_nRF52832\_mesh\_server\_provisioned.hex

1. Copy the meshfwpatch.py script to the <Mesh SDK 3.2.0>/scripts/interactive\_pyaci/ directory. The script has been tested to run only from this directory.
2. Look in the <Mesh SDK 3.2.0>/scripts/interactive\_pyaci/database/example\_database.json file and note that the “node” section has only one entry under it:

"nodes": [

{

"UUID": "466d9fd62df45244998a2bbd87936f35",

"appKeys": [

0,

1

],

"cid": "0059",

"configComplete": false,

"crpl": "0028",

"deviceKey": "8b9f87d7c2e13c160194805fe56b9aef",

"elements": [

{

"index": 0,

"location": "0000",

"models": [

{

"modelId": "0000"

},

{

"modelId": "0002"

},

{

"bind": [

0

],

"modelId": "1000"

}

]

}

],

"features": {

"friend": 2,

"lowPower": 2,

"proxy": 0,

"relay": 0

},

"name": "Light bulb",

"netKeys": [

0

],

"pid": "0000",

"security": "low",

"unicastAddress": 16,

"vid": "0000"

}

],

This implies an index of 0 and it holds the device key that we need to search for in the ripped provisioned firmware.

Type in the following command to generate the new firmware:

python meshfwpatch.py --hex-input-file dump\_nRF52832\_mesh\_server\_provisioned.hex --db-input-file database\test\_database.json --hex-output-file nRF52832\_mesh\_server\_copy\_1.hex

1. The generated “nRF52832\_mesh\_server\_copy\_1.hex” file can now be flashed to a new node. After flashing to new node using nrfjprog, switch the node off and then on and it should come up already pre-provisioned.

NOTES:

* For different types of nodes in a mesh network, the first node needs to be manually provisioned and then it can be processed with the script to generate firmware for other nodes of the same type. For instance, if there is a light switch client node in the mesh network, the first client node is manually provisioned, and then other client nodes can reuse the firmware ripped from that client node after processing by the script.
* Use the -h parameter to have meshfwpatch.py script display a help screen.
* The device key and the unicast address are automatically generated but can be specified on the command line as well.
* The optional parameters to the script are -–log-level, --node-name, --unicast-addr, -–device-key, and -–start-node.